Improving Indian Language Dependency Parsing by Combining Transition-based and Graph-based Parsers

B. Venkata Seshu Kumari  
Associate professor  
Dept of CSE  
St. Peter’s Engineering College,  
Hyderabad, India

R. Rajeswara Rao  
Associate Professor  
Dept of CSE  
JNTU college of Engineering, Vijayanagaram  
JNTUK, India

ABSTRACT
We report our dependency parsing experiments on two Indian Languages, Telugu and Hindi. We first explore two most popular dependency parsers namely, Malt parser and MST parser. Considering pros of both these parsers, we develop a hybrid approach combining the output of these two parsers in an intuitive manner. For Hindi, we report our results on test data provided in the for gold standard track of Hindi Shared Task on Parsing at workshop on Machine Translation and parsing in Indian Languages, Coling 2012. Our system secured unlabeled attachment score of 95.2% and labelled attachment score 90.7%. For Telugu, we report our results on test data provided in the ICON 2010 Tools Contest on Indian Languages Dependency Parsing. Our system secured unlabeled attachment score of 92.0% and labelled attachment score 69.5%.

Keywords  
Dependency Parsing; Telugu; Hindi; Malt Parser; MST Parser

1. INTRODUCTION
Dependecy parsing is the task of uncovering the dependency tree of a sentence, which consists of labeled links representing dependency relationships between words. Parsing is useful in major NLP applications like Machine Translation, Dialogue systems, text generation, word sense disambiguation etc. This led to the development of grammar-driven, data-driven and hybrid parsers. Due to the availability of annotated corpora in recent years, data driven parsing has achieved considerable success. The availability of phrase structure treebank for English has seen the development of many efficient parsers.

Unlike English, many Indian (Hindi, Bangla, Telugu, etc.) languages are free-word-order and are also morphologically rich. It has been suggested that free-word-order languages can be handled better using the dependency based framework than the constituency based one (Bharati et al., 1995). Due to the availability of dependency treebanks, there are several recent attempts at building dependency parsers. Two CoNLL shared tasks (Buchholz and Marsi, 2006; Nivre et al., 2007b) were held aiming at building state-of-the-art dependency parsers for different languages. Recently in We first explored Malt and MST parsers for parsing Telugu and Hindi. Considering pros of both these, two ICON Tools Contest (Husain, 2009; Husain et al., 2010), rule-based, constraint based, statistical and hybrid approaches were explored. Zeman (2009), and Kosaraju et al. (2010) used Malt Parser and explored the effectiveness of local morphosyntactic features, chunk features and automatic semantic information. Parser settings in terms of different algorithms and features were also explored. Zaman (2009) combined various well known dependency parsers forming a super parser by using a voting method. Yeleti and Deepak (2009) and Kesedi et al. (2010) used a constraint based approach. The scoring function for ranking the base parses is inspired by a graph based parsing model and labeling. Attardi et al. (2010) used a transition based dependency shift reduce parser (DeSR parser) that uses a Multilayer Perceptron (MLP) classifier with a beam search strategy.

2. RELATED WORK
In two ICON Tools Contest (Husain, 2009; Husain et al., 2010), different rule-based, constraint based, statistical and hybrid approaches were explored towards building dependency parsers for Indian languages. Ghosh et al. (2009) used a CRF based hybrid method. Nivre (2009), Ambati et al. (2009), and Kosaraju et al. (2010) used Malt Parser and explored the effectiveness of local morphosyntactic features, chunk features and automatic semantic information. Parser settings in terms of different algorithms and features were also explored. Zeman (2009) combined various well known dependency parsers forming a super parser by using a voting method. Yeleti and Deepak (2009) and Kesedi et al. (2010) used a constraint based approach. The scoring function for ranking the base parses is inspired by a graph based parsing model and labeling. Attardi et al. (2010) used a transition based dependency shift reduce parser (DeSR parser) that uses a Multilayer Perceptron (MLP) classifier with a beam search strategy.

3. APPROACH
We explored two data-driven parsers Malt parser (Nivre et al., 2007a), and MST parser (McDonald et al., 2006) for our experiments in this paper. In this section, we first describe both these two parsers in detail. Then we explain our approach of combing these two parsers to produce better parser output for Telugu and Hindi.

4. MALT PARSER
Malt parser is a freely available implementation of the parsing models described in (Nivre et al., 2007a). Malt parser implements the transition-based approach to dependency parsing, which has two essential components:
7. **TELUGU: DATA AND SETTINGS**

7.1 **Data**

For our experiments, we used Telugu data from ICON 2010 Tools contest. Data released has both fine-grained and coarse-grained versions of dependency labels. We used fine-grained version here. This data was annotated using the Computational Paninian Grammar (Bharati et al., 1995). The annotation scheme based on this grammar has been described in Begum et al. (2008) and Bharati et al. (2009). Subject and direct object equivalent dependency in this framework are kartha karaka (k1) and karma karaka (k2). Table 1 shows the training, development and the testing data sizes the Telugu treebank. Statistics on sentence count, word count and average sentence length are provided in this table.

### Table 1 – Telugu treebank statistics

<table>
<thead>
<tr>
<th>Type</th>
<th>Sent Count</th>
<th>Word Count</th>
<th>Avg. sent_length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train</td>
<td>1,400</td>
<td>7602</td>
<td>5.43</td>
</tr>
<tr>
<td>Devel</td>
<td>150</td>
<td>839</td>
<td>5.59</td>
</tr>
<tr>
<td>Test</td>
<td>150</td>
<td>836</td>
<td>5.57</td>
</tr>
</tbody>
</table>

7.2 **Parser Settings**

As the training data size is small, we merged training and development data and did 10-fold cross validation for tuning the parameters of the parsers and for feature selection. Best settings obtained using cross-validated data are applied on test set. In case of Malt parser, liblinear learner and arc-eager parsing algorithm consistently gave better performance. For feature model, we tried best feature settings of the same parser on different languages in CoNLL and ICON shared tasks (Hall et al., 2007; Husain 2009; Husain et al., 2010) and applied the best feature model.

In case of MST, order=2, training-k=1 and non-projective algorithm gave the best results. It was difficult to do feature tuning with MST parser as it do not provide nice options similar to Malt parser. We explored different features in labelling module of the MST parser and selected the settings which gave best results on 10-fold cross-validation.

8. **HINDI: DATA AND SETTINGS**

8.1 **Data**

We used gold standard track of Hindi Shared Task on Parsing at Coling 2012 MTIPIL workshop. Similar to Telugu, this data was annotated using the Computational Paninian Grammar (Bharati et al., 1995). The annotation scheme based on this grammar has been described in Begum et al. (2008) and Bharati et al. (2009). Subject and direct object equivalent dependency in this framework are kartha karaka (k1) and karma karaka (k2). We explored different features provided in the FEATS column and found that only root, category, vibhakti, TAM and chunk information are useful. Gender, number, person and other information didn’t give any improvements. This observation is similar to previous work by Ambati et al. (2010) and Kosaraju et al. (2010). Table 2 shows the training, development and the testing data sizes the Telugu treebank. Statistics on sentence count, word count and average sentence length are provided in this table.
8.2 Parser Settings
For Malt, we explored different parser algorithms for Hindi and found that nivre arc-standard gave better performance over others. In case of learning algorithms, LIBLINEAR gave better performance compared to LIBSVM. Also, LIBLINEAR was very faster than LIBSVM learner.

In case MST, we different options provided by the parser and found that non-projective algorithm and training-k=5, gave best results.

<table>
<thead>
<tr>
<th>Type</th>
<th>Sent Count</th>
<th>Word Count</th>
<th>Avg. sent_length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train</td>
<td>12,041</td>
<td>268,093</td>
<td>22.27</td>
</tr>
<tr>
<td>Devel</td>
<td>1,233</td>
<td>26,416</td>
<td>21.42</td>
</tr>
<tr>
<td>Test</td>
<td>1,828</td>
<td>39,775</td>
<td>21.76</td>
</tr>
</tbody>
</table>

9. EXPERIMENTS AND RESULTS
Performance of Malt parser and MST parser on test data are provided in Table 3. We used standard Labelled Attachment Score (LAS), Un-labeled Attachment Score (USA) and Labeled Score (LS) metrics for our evaluation.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Telugu</th>
<th>Hindi</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UAS</td>
<td>LAS</td>
</tr>
<tr>
<td>Malt</td>
<td>91.8%</td>
<td>70.0%</td>
</tr>
<tr>
<td>MST</td>
<td>90.0%</td>
<td>67.1%</td>
</tr>
<tr>
<td>Our Approach</td>
<td>92.0%</td>
<td>69.5%</td>
</tr>
</tbody>
</table>

Table 4 – Performance of Malt parser, MST parser and our approach on top five dependencies in the test data.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Telugu</th>
<th>Hindi</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Malt Pars er</td>
<td>MST Pars er</td>
</tr>
<tr>
<td>Main</td>
<td>97.0</td>
<td>95.3</td>
</tr>
<tr>
<td>k1</td>
<td>63.0</td>
<td>59.4</td>
</tr>
<tr>
<td>k2</td>
<td>58.8</td>
<td>62.7</td>
</tr>
<tr>
<td>ccof</td>
<td>83.1</td>
<td>74.4</td>
</tr>
<tr>
<td>r6</td>
<td>81.5</td>
<td>53.9</td>
</tr>
</tbody>
</table>

9.1 Analysis: Telugu
On the test data, Malt parser and MST parser gave UAS of 91.8% and 90.0% respectively. Using our approach, we could achieve UAS of 92.0%, which is better than both the baseline systems. Similarly, Malt parser and MST parser gave LAS of 70.0% and 67.1% respectively. Our approach gave an LAS of 69.5%. Though it is slightly lower than Malt parser’s performance, it is much higher than the MST parser’s performance. As performance of MST parser is much lower compared to Malt, there is only slight improvement in case of UAS and slight decrement in case of LAS. We hope that if we can improve MST parser's performance then we could achieve much better improvements with our approach. As the training data is very low, and also as Telugu is agglutinative language, LAS for the all the systems is very low. With more training data and specialized techniques for handling agglutinative languages like Telugu, we can achieve better results in LAS.

Table 4, gives an overview of the performance of Malt parser, MST parser and our approach on the top five dependencies. Results show that our approach outperforms both Malt parser and MST parser on major dependencies. We couldn't get much improvement in case of k2. We believe this could be because of low performance of MST parser. By obtaining similar performance on short distance dependencies and huge improvements on long distance dependencies (by taking MST output) over Malt, we could achieve better accuracies over both the parsers. Taking the fact that Malt parser is good at short distance dependencies and MST parser is good at long distance dependencies, into consideration, we developed our system, which outperformed both Malt and MST parsers.

9.2 Analysis: Hindi
On the test data, Malt parser and MST parser gave UAS of 93.9% and 95.8% respectively. Using our approach, we could achieve UAS of 95.2%, which is better than Malt but slightly lower than Malt. Malt parser and MST parser gave LAS of 89.4% and 89.2% respectively. Our approach gave an LAS of 90.7% which is better than both the baselines. As performance of Malt parser is much lower compared to MST, there is only slight decrement in case of UAS. We hope that if we can improve Malt parser’s performance then we could achieve much better improvements with our approach.

Table 4, gives an overview of the performance of Malt parser, MST parser and our approach on the top five dependencies. Results show that our approach outperforms Malt parser in all the cases and MST parser on few dependencies. We couldn't get much improvement in case of main and k1 as MST is far better at these labels compared to Malt. By obtaining similar performance on short distance dependencies and huge improvements on long distance dependencies (by taking MST output) over Malt, we could achieve better accuracies over both the parsers. Taking the fact that Malt parser is good at short distance dependencies and MST parser is good at long distance dependencies, into consideration, we developed our system, which outperformed both Malt and MST parsers.

10. CONCLUSION AND FUTURE WORK
In this paper, we first explored Malt and MST parsers and developed best models, which we considered as the baseline models for our approach. Considering pros of both these parsers, we developed a hybrid approach combining the output of these two parsers in an intuitive manner. As Malt parser is good at short distance dependencies and MST parser is good at long distance dependencies, we gave more
weightage to Malt parser in case of short distance dependencies and gave more weightage to MST parser in case of long distance dependencies. We showed that a simple system like combining both MST and Malt parsers in an intuitive way, can perform better than both the parsers. For Hindi, we reported our results on test data provided in the for gold standard track of Colling 2012 MTPIL workshop. Our system secured unlabeled attachment score of 95.2% and labelled attachment score 90.7%. For Telugu, we report our results on test data provided in the ICON 2010 Tools Contest on Indian Languages Dependency Parsing. Our system secured unlabeled attachment score of 92.0% and labelled attachment score 69.5%.

In our current approach, we combined the output of both Malt and MST parsers to get a better system over both the parsers. In future, we would like to combine both the models in a way similar to McDonald and Nivre (2007). We also would like to explore the approach of voting similar to Zeman (2009) by taking advantages of different available parsers. We also plan to explore the usefulness of large un-annotated data using self-training and co-training techniques to improve the performance of the Indian Language dependency parsers.

10.1 Acknowledgment
We would like to thank Language Technologies Research Institute (LTRC), International Institute of Information Technology, Hyderabad (IIIT-H) for providing the data and previous best settings for the parser.

11. REFERENCES


